Acceso a Datos

UT02- Herramientas ORM

Act.01 – APP STOCK CONTROL

Prof. [Diego Linares Ortiz]

UT02. Act01
App Stock Control

Objetivo

El objetivo del proyecto es que la aplicación quede funcionando, es decir, que, desde el main, cuando se pulse cada una de las opciones, se obtenga el resultado esperado de cada opción.

- Ejemplo: Desde el main se pulsa en la opción 1 (alta producto) -> El resultado esperado sería que dicho producto se quede insertado correctamente dentro de la base de datos.
- Ejemplo2: Desde el main se pulsa en la opción 9 (obtener todos los proveedores) -> El resultado esperado es que se muestre un listado con todos los proveedores de la base de datos.

Especificaciones

Modelo

La capa Model define las estructuras de datos que representan las entidades principales de la aplicación. Estas entidades suelen estar mapeadas directamente a tablas de la base de datos o a documentos en bases de datos NoSQL. Ejemplos típicos son clases como User, Product, Order, que corresponden a usuarios, productos o pedidos. Definen los atributos y las relaciones de las entidades. Especifican restricciones y reglas de validación (como tipos de datos, restricciones de unicidad, valores por defecto).

En esta base de datos tenemos 3 tablas:

1. **Tabla Proveedores**
   - id: Long -> AutoGenerado. PrimaryKey
   - nombre: String -> único, longitud 50, not null
   - direccion: String -> not null
   - productos: List -> Relacion de @OneToMany
2. **Tabla Productos**
   - id: String -> PrimaryKey -> Lo generamos nosotros
   - categoria: String -> not null, longigud 10
   - nombre: String -> longitud 50, not null
   - descripcion: String
   - precio_sin_iva: float -> not null
   - precio_con_iva: float -> not null

- fecha_alta: Date -> not null
- stock: int -> not null
- proveedor: Proveedor -> Relación de @ManyToOne

3. **Tabla Usuarios**
   - nombre_usuario: String -> PrimaryKey
   - password: String -> not null, longitud 20

Service

La capa de **Service** encapsula la **lógica de negocio** de la aplicación. Es el puente entre los controladores y los repositorios, asegurando que todas las reglas de negocio y procesos complejos se gestionen de manera centralizada y coherente. Además, esta capa promueve la reutilización de código, ya que la lógica puede ser usada por diferentes controladores. La lógica de negocio para cada Entidad es la siguiente:

1. **Proveedor**
   - nombre: único, longitud 50, not null
   - direccion: not null
2. **Producto**
   - id: El id se compone por: 3 primeras letras de categoria + 3 primeras letras de nombre + 3 primeras letras de proveedor
   - categoria: longitud 50, not null
   - nombre: longitud 50, not null
   - precio_sin_iva: not null
   - precio_con_iva: not null. Calcular el precio aplicando el IVA sobre el precio sin iva
   - fecha_alta: fecha de hoy

Repository

En este paquete recordad que debéis implementar las clases que acceden a la base de datos.

En este caso, tenemos una base de datos MySQL gestionada por Hibernate.

En esta base de datos tenemos 3 tablas:

1. Tabla Proveedores
2. Tabla Productos
3. Tabla Usuarios

Hay que crear al menos 3 clases Repository, una por cada Entidad que queremos manejar de la base de datos.

Explicación de métodos para la gestión de productos

1. **altaProducto**: Este método está diseñado para registrar un nuevo producto. Los parámetros incluyen detalles sobre el producto (como su ID, nombre, precio sin IVA, descripción) y sobre el proveedor (ID, nombre, dirección). La respuesta de este método debería reflejar si la operación de alta fue exitosa o no.
    - Parámetros:
        - idProducto: Identificador único del producto.
        - nombreProducto: Nombre del producto.
        - precioSinIva: Precio del producto sin incluir el IVA.
        - descripcionProducto: Descripción del producto.
        - idProveedor: Identificador único del proveedor.
        - nombreProveedor: Nombre del proveedor.
        - direccionProveedor: Dirección del proveedor.
2. **bajaProducto**: Este método realiza la baja (eliminación) de un producto, dada su ID. La respuesta debería contener el resultado de la operación, es decir, si el producto fue eliminado exitosamente o si ocurrió algún error.
    - Parámetro:
        - id: Identificador del producto a eliminar.
3. **modificarNombreProducto**: Este método permite modificar el nombre de un producto existente. Se requiere el ID del producto para identificarlo y el nuevo nombre que se le asignará. La respuesta debería reflejar el éxito o fallo de la modificación.
    - Parámetros:
        - id: Identificador del producto cuyo nombre se desea cambiar.
        - nuevoNombre: Nuevo nombre que se asignará al producto.
4. **modificarStockProducto**: Permite modificar la cantidad de stock de un producto específico. El método recibe el ID del producto y el nuevo valor del stock. La respuesta indicará si la actualización fue exitosa.
    - Parámetros:
        - id: Identificador del producto cuyo stock será modificado.
        - nuevoStock: Nuevo valor del stock del producto.

5. **getProducto**: Este método busca y devuelve la información de un producto en particular, identificado por su ID. La respuesta contendrá los detalles del producto si la búsqueda fue exitosa.
    - Parámetro:
        - id: Identificador del producto a obtener.
6. **getProductosConStock**: Este método busca y devuelve una lista de todos los productos que tienen stock disponible. La respuesta incluirá una lista de productos con stock mayor a cero.
7. **getProductosSinStock**: Este método devuelve una lista de productos que no tienen stock disponible (su stock es igual a cero). La respuesta incluirá una lista de productos que están agotados.

Explicación de métodos para la gestión de proveedores

1. **getProveedorProducto**: Este método está diseñado para obtener el proveedor asociado a un producto específico, identificado por su ID. El método devuelve el proveedor que suministra dicho producto. La respuesta debe incluir el proveedor si la operación fue exitosa.
    - Parámetro:
        - idProducto: Identificador del producto del cual se desean obtener el proveedor.
2. **getTodosProveedores**: Este método devuelve una lista de todos los proveedores registrados en el sistema. No requiere parámetros y su resultado será la lista completa de proveedores disponibles. La respuesta debe reflejar si la operación fue exitosa y contener los datos de los proveedores.

Explicación del método para iniciar sesión

login: Este método se utiliza para autenticar a un usuario. Recibe como parámetros el nombre de usuario o identificación (userInput) y la contraseña (passInput). El objetivo del método es validar las credenciales ingresadas por el usuario y, si son correctas, devolver una respuesta HTTP con la información del usuario autenticado.

- Parámetros:
    - userInput: Identificador del usuario (puede ser el nombre de usuario, correo electrónico u otro identificador).
    - passInput: Contraseña del usuario para autenticación.

La respuesta del método debe contener el resultado de la autenticación: si el usuario ha iniciado sesión correctamente, se devolverá la información correspondiente al usuario; de lo contrario, se indicará un fallo en la autenticación.

Realice las siguientes tareas:

Desarrolla una pequeña aplicación que cumpla con las especificaciones dadas y que esté separada en las diferentes capas lógicas.

RAs y CEs evaluados

RA 3. Gestiona la persistencia de los datos identificando herramientas de mapeo objeto relacional (ORM) y desarrollando aplicaciones que las utilizan.

a) Se ha instalado la herramienta ORM.

Indicadores	Niveles de logro		
	Bien	Regular	Insuficiente
Indica cómo instalar la herramienta ORM	100%	50%	0%

b) Se ha configurado la herramienta ORM.

c) Se han definido los ficheros de mapeo.

Indicadores	Niveles de logro		
	Bien	Regular	Insuficiente
Define ficheros de mapeo	100%	50%	0%

d) Se han aplicado mecanismos de persistencia a los objetos.

e) Se han desarrollado aplicaciones que modifican y recuperan objetos persistentes.

Indicadores	Niveles de logro		
	Bien	Regular	Insuficiente
Desarrolla una aplicación según las especificaciones	100%	50%	0%
Organiza la aplicación en capas	100%	50%	0%

f) Se han desarrollado aplicaciones que realizan consultas usando el lenguaje SQL.

Indicadores	**Niveles de logro**		
	Bien	**Regular**	**Insuficiente**
Utiliza JPQL para consultas complejas	100%	50%	0%

g) Se han gestionado las transacciones.

Indicadores	**Niveles de logro**		
	Bien	**Regular**	**Insuficiente**
Gestiona las transacciones	100%	50%	0%